The UbiquiCoin Consensus Protocol: Proof of Majority
A Democratic Model for Decentralized Network Consensus

JEFF MAHONY: Co-Founder and CEO of UbiquiCoin LLC
Draft 1.0 - November 12, 2017

Table of Contents

1. Abstract

The majority consensus model introduced in this paper (“Proof of Majority”), addresses the latency issues, centralization issues, and non-democratized qualities of the prevailing blockchain technologies, particularly those underlying the cryptocurrencies, available as of the time of the writing of this paper. Current Proof of Work, Proof of Stake, and Proof of Authority consensus models and any consensus models that rely on the selection of a single node to produce, and ultimately extend for validation, a block during a blockchain epoch are not useful in real world applications that require low latency processing and small footprint processing power. Further, these consensus models don’t meet their stated requirements of decentralization and democratization. Instead, they fall far short of their original intents. In each model, a single node is selected via some mechanism, whether by “lottery” as with Proof of Work, by random selection among the largest coin holders as with Proof of Stake, or by random selection among nodes deemed worthy by a centralized decision-making body, as with Proof of Authority. Regardless of how the single node is selected, the single node becomes a defacto-authority voting for the entire network. This is not democratization. Additionally, and although unintended, each of these models is ultimately run by a defacto-central authority and not by the network as a set of decentralized nodes. Due to the arbitrarily difficult work required to be performed to “prove” that the block generated by the single selected node is worthy of network wide validation, massive processing power is required. Thus, a small group of large pools of processing power have risen to the status of defacto-authorities in the case of Proof of Work based blockchains. Proof of Stake is by its very nature designed to concentrate power in the hands of a few based on their wealth, and Proof of Authority leaves the decision of which entities can operate nodes in the network to a central vetting authority. Thus, each of these models has fallen short of their stated requirements.

The Proof of Majority consensus model delivers a decentralized and democratized blockchain with low latency, low processing power requirements, low network traffic, low network energy, frictionless and guaranteed transaction processing, attack vector mitigation, and mining rewards.

During a given *epoch*, all nodes in the network are responsible for building their "version" of a block containing all the current unconfirmed transactions. No node is required to perform arbitrary hashing work in order to be selected as the dominant miner; who is ultimately allowed to present its block for validation to the rest of the network through costly network *gossiping* protocols involving large amounts of information. Instead, all nodes generate a block, whose hash they pass, only, to each of their connected peers. The hashes are then validated and compared to determine a majority consensus within the connected peer group. Once each node arrives at a majority decision within its peer group, the node adds the block to its understanding of the network wide blockchain. Because all the nodes are building their "version" of the block, which can be validated and compared to other blocks within the node's directly connected peer group, all nodes must start with the same set of transactions. This is done by propagating the unconfirmed transaction lists through the network to converge on the longest list that will fit into a block. A consensus of which transactions to include is arrived at before a block consensus is arrived at. All validated transactions are included; none are skipped or ignored simply because the associated fee is not interesting to miners. Nodes do not place a "coinbase" transaction into the block they are building; no dilution of coin value through mining is allowed. Doing so will doom the entire block constructed by the node to later be deemed by its peers as a "minority" block and it will be eliminated. Additionally, there is no need for senders to allocate a fee in their transaction request, as these types of incentives are ignored by nodes building blocks. All transactions are included in block building without fees.

In the Proof of Work consensus model, arbitrary workloads at each node in the network, cause huge prolonged levels of network energy. Regardless of who wins the "lottery" and is "allowed" to submit their block for validation, the entire network was attempting to win and thus there is a network wide rise in energy consumption. In contrast, in the Proof of Majority consensus model, network energy remains ultra-low. Nodes are simply compiling a block of transactions and doing very little hashing work; work that is done in milliseconds, not minutes. Thus, nodes in the Proof of Majority network can be very light weight with an extremely low processing footprint and a very small memory footprint. Data movements are never much larger than the size of a block and movement frequency is naturally throttled by transaction list formation and consensus building activities on each node. Hardware running nodes can be a fraction of the processing power required to run nodes in a Proof of Work consensus model. The Proof of Majority nodes can run on handheld devices; smart phones have the necessary processing power, memory, and bandwidth.

2. Honest Nodes

As with all decentralized and democratized network computers, *honest* nodes must outnumber *dishonest* nodes to maintain the probability that the distributed computer will not be compromised. A fully connected peer-to-peer network is the ideal way to mitigate the impact of *dishonest* nodes in this type of computer. Unfortunately, network packet delivery issues, node processing and memory limits, and latency issues, prevent the ideal network at scale. Typical world-wide networks of this nature will on average have peer groups of 8-10 nodes; this is generally a defensible number from a probability standpoint. At their infancy, however, these types of networks will have less nodes in a peer group and there will be less peer groups, making them particularly vulnerable to *dishonest* nodes and other attack vectors during this time. In any network state, *honest* nodes follow the rules and create an orderly path to solving the problem at hand. When the problem at hand is generating valid blocks, and adding them to the valid blockchain as understood by the network at large, the more connected direct peers a node has and the larger the overall node network is, the more likely the computer can survive *dishonest* nodes; invalid forks are more quickly abandoned, and thus *dishonest* nodes have to work harder and there needs to more of them. However, since the network isn't always privy to which nodes are *dishonest*, even *honest* nodes have to be throttled to ensure that their data is not corrupted by a *dishonest* node and introduced into the network as valid data. To do this, the Proof of Majority consensus model "timelocks" the first node in each peer group to come up with a valid block; effectively ignoring the node's block generations in the consensus for a certain number of blocks. As a result, at any given moment, a small portion of the network's *honest* nodes are being ignored, but at the advantage of mitigating *honest* node corruption by *dishonest* nodes' injection of data that, though validates, does not meet the majority consensus; invalid forks are abandoned very quickly in this model.

3. Dishonest Nodes

Decentralized and democratized network computers are open to many attack vectors; denial of service, Sybil attacks, packet sniffing, forcing clock drift attacks, security vulnerabilities, and the like. But the most destructive attack vector is the dissemination of illegal information by seemingly *honest* contributors; information that though can be validated is nevertheless erroneous. *Dishonest* nodes attack peer groups, intending to infiltrate the peer group and get the block they generate into the blockchain. Because the majority of blocks given to a node by its connected active peers must be valid and must match before a majority consensus is arrived at, *dishonest* nodes would have to control the

peer group to affect a negative outcome. Fortunately, peer groups are not isolated islands in the network, they themselves are connected to other peer groups and so on. This makes the problem of the *dishonest* node statistically more difficult. So *dishonest* nodes would ultimately have to overwhelm the network in terms of numbers to sway a network wide majority decision model. Unfortunately, with today's computing resources, this is not completely unattainable. So, more precautions have to be taken. Proof of Work does a good job of mitigating this attach vector through its "lottery" system of node selection, forcing *dishonest* nodes to not only outnumber the *honest* nodes, but to be able to perform more work than the collective pool of *honest* nodes. Proof of Majority mitigates this risk, by "blacklisting" any node whose block does not match the other blocks being evaluated by any given node. This also means that any *honest* node that has been overwhelmed by *dishonest* nodes will soon be "blacklisted" by the network, thereby, mitigating the risk of *dishonest* nodes creating forks that aren't quickly abandoned. "Blacklisting" is a great tactic, but with one glaring fault; *dishonest* nodes simply have to work to get the majority of the *honest* nodes "blacklisted". To prevent this, the Proof of Majority model allows nodes to come off the "blacklist" after a certain number of blocks have been created; statistically significant time to abandoned corrupt forks, while allowing *honest* nodes to re-enter the work force.

4. Wallets and Addresses

Proof of Majority uses elliptic curve cryptography to ensure confidentiality, authenticity, and non-repudiability of all transactions originated within wallets. Each wallet is a private/public Ed25519. Wallets are identified by addresses, which are derived in part from one-way mutations of Ed25519 public keys.

4.1 Wallet Address

A wallet address is a base-323 encoded triplet consisting of; (1) network byte, (2) 160-bit hash of the wallet's public key, and (3) 4 byte checksum to allow for quick recognition of mistyped addresses.

4.2 Generating an Address

To convert a public key into a wallet address, the following steps are performed:

01. Perform 256-bit Sha3 on the public key
02. Perform 160-bit Ripemd of hash resulting from step 01.
03. Prepend version byte to Ripemd hash (either 0x68 or 0x98)
04. Perform 256-bit Sha3 on the result, take the first four bytes as a checksum
05. Concatenate output of step 03 and the checksum from step 04
06. Encode result using base32

5. Cryptography

Proof of Majority requires the use of cryptography in all aspects of internode communications and is based on Elliptic Curve Cryptography using the Twisted Edwards Curve to help ensure security, speed, and low processing requirements, as follows:

$$-x^2 + y^2 = 1 - \frac{121665}{121666}x^2y^2$$

over the finite field defined by the prime number $2^{225} - 19$ together with the digital signature algorithm called Ed25519 [1]. The base point for the corresponding group G is called B. The group has $q = 2^{252} - 27742317777372353535851937790883648493$ elements. Every group element A can be encoded into a 256-bit integer $\underline{A}$ which can also be interpreted as 256-bit string and $\underline{A}$ can be decoded to receive A again.

For strict hashing, the consensus mode uses the 512-bit SHA3 hash function.

5.1 Encrypted Message Traffic

Node communications in the clear is just an open door to many attack vectors. This consensus model dictates that nodes communicate using only encrypted messages. Virtually all encryption algorithms are subject to breakage, but the pool of bad actors capable of launching attacks is significantly reduced. Rotating keys and node identification protocols further reduce that pool. Elliptic Curve Cryptography is compact and fast and won't significantly add to the node footprint or to node latency. Key security is a major factor in the successful use of encryption. Proof of Majority provides solutions for key protection and node identification to mitigate spoofing and virus/worm activities that steal keys. Solutions are also in place to mitigate replay attacks. Peer connection attempts must use the proper encryption and handshake protocols to successfully join a peer group. Because, nothing can completely eliminate these kinds of risk in a distributed computer, this consensus model employs strategies to reduce the pool of bad actors capable of launching attacks; message encryption is a key strategy.

6. Transactions

Wallet states can only be effected through transactions confirmed onto the blockchain. Wallet owners create transactions and send them to their connected nodes. These transactions are known as *unconfirmed transactions*. Unconfirmed transactions are processed by the network of nodes for no fee; consensus is frictionless. State migration from unconfirmed to confirmed for a given transaction can only be accomplished through a majority consensus of the entire network of nodes and is represented by the *election* of a new block containing the given transaction.

When a wallet creates a new transaction, the transaction is broadcast to all connected active peers. The receiving nodes validate the transaction and, if valid, the nodes place the transaction into their *unconfirmedtransactions* queue. At the next *epoch,* the node then *gossips* the transaction to the network at large via the *transaction propagation* protocol. Prior to processing unconfirmed validated transactions into a block, the network establishes a complete list through the *transaction balloting* protocol to ensure that the network has converged.

7. Node Processing

The blockchain represents the entire state of the network of nodes at any given point in time. Blocks on the blockchain represent a consensus among the nodes in the network reflecting a network wide accepted statement about the state of the network at the time the block was added to the blockchain.

In any consensus model operating in a decentralized network, one or more nodes on the network assert a statement or statements about the state of the network to other nodes for consideration by those other nodes as to the accuracy of that statement or statements. In the Proof of Majority consensus model, all nodes are asserting one statement each about the state of the network, at that time, in the form a block.

Thus, the primary purpose of a node is to arrive at a representation of the state of the network that is accurate. It must do this without knowledge of the entire network and in a decentralized manner, while mitigating attack vectors, particularly bad actors – i.e., *dishonest* nodes.

Nodes must first become an integral part of the network, discovering other nodes and then establishing vectors of communications with those nodes. Nodes are not required to establish communication vectors with all other nodes, to avoid unnecessary network traffic and heightened network wide energy levels. Instead, nodes form or join groups that are connected to other groups through nodes common to the groups. In doing so, a network is established. As new nodes enter the network, node peering becomes more resilient and more resistant to attack vectors. In the Proof of Majority consensus model, all nodes participate in forming the consensus of the network state.

7.1 Peer Sampling

Every node in the network requires peers to connect to for the purposes of exchanging messages about the state of the network. However, every node does not have to exchange messages with every other node in the network to have the probability of complete information sharing among nodes approach 1. In practice, it is not feasible for every node to exchange message with every other node in a large network, where nodes regularly join and leave the network, either purposefully or because of failure, and where the network suffers from broken or slow links, as is the case in a truly decentralized network, where independent operators maintain nodes. Instead, it has been shown that nodes should exchange messages with peers that are selected following a uniform random sample of *all nodes* currently in the network [2][3]. Based on this assumption, it is possible to achieve high levels of scalability, reliability, and efficiency. However, enforcing a network wide uniform random sampling of *all nodes* requires that every node knows every other node in the network [4]. This is an unrealistic goal due to the considerable synchronization costs that would otherwise be incurred in a decentralized peer-to-peer network where individual nodes are often connected for only a few minutes to an hour (*high churn rates*), as various measurement studies show [5][6]. Instead, it has been shown that using the *gossip* paradigm, nodes can construct, and refresh, at each *epoch*, a random sampling of peers, sufficient to create a probability of complete information sharing among nodes that approaches 1, from a localized partial view of the complete set of network nodes, without having to know the complete set of network nodes [7].

In the Proof of Majority consensus model, an efficient *peer sampling* protocol is crucial to staging and conducting the consensus mechanism's supporting protocols. The goal of the *peer sampling* protocol is to provide each node with a random subset of peers from the peer groups it is connected to directly or indirectly. Each node in the network will construct and maintain entries in a *randompeers* queue, such that the queue contains one and only one descriptor for each peer node contained in the queue. Each descriptor contains a network address for the peer node and an age that represents the freshness of the given node

descriptor. The queue must maintain a number of *node descriptors* (c), such that $c : O(log\ V)$ where V is the number of connected nodes in the network [8][7], to be an effective base to the *gossip* protocols used in other node protocols. If this is the network start or node start then each node sets its *randompeers* queue to its set of connected active peers and initializes their age to zero.

To continually reflect the dynamics of the network, peer sampling is assumed to be executed only once per *epoch*, by each node n, using the following schemas centered on a push/pull approach:

Node n performs a single *randompeers* queue exchange with a peer p during an *epoch*:

01. Select p as the highest aged descriptor in the *randompeers* queue that represents an *active* peer and that is not in the *suspectednode* queue
02. Package a buffer b as follows:
    a. Add descriptor for node n to b
    b. Shuffle the *randompeers* queue and move the highest aged descriptor to the end of the *randompeers* queue
    c. Append the first $c/2 - 1$ descriptors in the *randompeers* queue to b
03. Send b to p and set answer flag f to 0 for p
04. Increase the *age* component of each descriptor in the *randompeers* queue by 1

Node n is receiving a buffer b_1 from another node p:

01. If p is not in the *suspectednode* queue, then
    a. If f is set to 1, then
        i. Package a buffer b_2 as follows:
            1. Add descriptor for node n to b_2
            2. Shuffle the *randompeers* queue and move the highest aged descriptor to the end of the *randompeers* queue
            3. Append the first $c/2 - 1$ descriptors in the *randompeers* queue to b_2
        ii. Send b_2 to p
    b. Update the *randompeers* queue, as follows:
        i. Append b_1 to the *randompeers* queue
        ii. Remove duplicates from the *randompeers* queue
        iii. Remove the highest aged descriptor from the *randompeers* queue
        iv. Remove the first min($c/2$, *randompeers* queue length – c) descriptors from the *randompeers* queue
        v. Remove at random (*randompeers* queue length – c) descriptors from the *randompeers* queue
    c. Increase the *age* component of each descriptor in the *randompeers* queue by 1
    d. Set answer flag f to 1 for p

By employing the two schemas above, nodes can reliably and efficiently refresh their sampling of localized peers to maintain a sufficiently randomized queue suitable for use in other node *gossip* based protocols. The peer sampling protocol ensures local randomness from each node's point of view and provides every node with *local* knowledge of the rest of system, which is the cornerstone to a network convergence of global properties using only local information [7]. Average path length is close to that characteristic of random graphs and there is high resilience to fault tolerance, particularly to high churn rates [7]. This protocol creates the opportunity for other node *gossip* based protocols to completely spread information across the network with a probability approaching 1 – i.e., the network converges. For this consensus model, c is set to 32, to affect a probability approaching 1 of saturating 4,294,899,255 nodes with information during *gossip* based protocols.

Since, the *randompeers* queue is updated only once per *epoch* and accepts a related information exchange from connected active peers only once per *epoch*, the problem of *dishonest* nodes controlling any given node's *randompeers* queue is mitigated. In the unlikely case, that two *honest* nodes collide, choosing each other to exchange peer information with, the node that receives the message first, ceases outbound information exchange during the *peer sampling* protocol.

7.2 Transaction Propagation

Rather than allowing nodes to determine which unconfirmed validated transactions are processed into a block, Proof of Majority mandates that each transaction, that is properly propagated and that can be validated, be processed into the nearest block possible. Transaction processing should be virtually guaranteed, mitigated only by catastrophic network failures, not *dishonest* nodes or other attack vectors. Crucial to approaching a guaranteed transaction processing system within the network, are protocols that can virtually ensure transaction propagation to all nodes.

Actual use of these protocols by nodes in the network is also crucial. The preponderance of current miner based systems do not have enough incentives for nodes to forward information at all, in their respective networks, because it is in the best interests of miners to hold on to transactions that include fees and claim those fees by eventually creating a block that includes the respective transactions [9]. Transaction processing should be devoid of fees to the parties involved in any given transaction, to incent usage of the system and to reduce system wide friction. The question of how nodes are incented to participate fully is addressed later in this paper.

A *transaction balloting* protocol is employed to ensure that prior to block generation, within an *epoch*, all nodes in the network are working with the same set of unconfirmed validated transactions in an ordered manner, with a certainty probability approaching 1. The *peer sampling* protocol is employed, within an *epoch*, to support the *transaction propagation* protocol and the *transaction balloting* protocol, among others. The *peer sampling* protocol is designed to keep the *randompeers* queue fresh and with sufficient peer descriptors to be suitable for random peer *gossiping*. *Transaction propagation* is handled primary through random peer *gossiping*.

Each node in the network will maintain entries in a *unconfirmedtransactions* queue, such that the queue contains one and only one descriptor for a given unconfirmed and validated transaction. The *unconfirmedtransactions* queue is populated through random peer *gossiping*, initiated first by the wallet node that created the transaction. Starting with the wallet node n, unconfirmed validated transactions are propagated, regardless of other processing being done by the node in a given *epoch*, using the following schemas centered on a push approach:

> Node n performs a single transaction t unidirectional exchange with d peers p as unconfirmed transactions are validated:
>
> 01. Select p as the d highest aged descriptors in the *randompeers* queue that represent *active* peers and that are not in the *suspectednode* queue
> 02. Package a buffer b with t
> 03. Send b to p
>
> Node n is receiving a buffer b containing transaction t from another node p:
>
> 01. If p is not in the *suspectednode* queue, then
>   a. If t is invalid, then add p to the *suspectednode* queue
>   b. If p is not in the *suspectednode* queue, then
>   i. If t is valid and does not exist in the *unconfirmedtransactions* queue for n, then
>   1. Update the *unconfirmedtransactions* queue with t
>   2. Perform a unidirectional exchange of t with d peers p, as follows:
>   a. Select p as the d highest aged descriptors in the *randompeers* queue that represent *active* peers and that are not in the *suspectednode* queue
>   b. Package a buffer b with t
>   c. Send b to p

It is not crucial that the network become saturated with a given unconfirmed validated transaction nor is it crucial to accommodate nodes entering the network post transaction propagation. Instead, *transaction balloting* will attempt to saturate the network prior to block creation. The goal of the *transaction propagation* protocol is only to mitigate the risk of a given transaction being *lost* or *dropped* by the network due to *high churn rates* common to decentralized peer-to-peer networks, where individual nodes are often connected for only a few minutes to an hour [5][6].

Consider the fifteen nodes in Figure 1, where the wallet node v_1 is asserting a transaction that will ultimately require the network at large to consider it for inclusion in a block during the *transaction balloting* protocol in the next *epoch*. Lines represent peer-to-peer connections in the network, through which messages reflecting the node's transaction are *gossiped* to other nodes in the network, devoid of any requirement to saturate the network. Instead, only nodes $\{v_1, \ldots, v_{15}\}$ receive the transaction information.

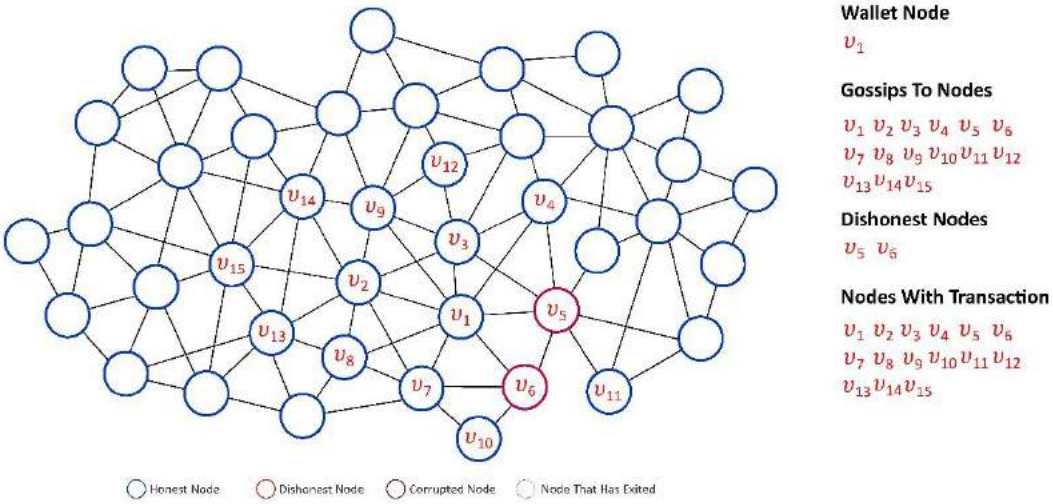

Figure 1. Wallet node v_1 *gossips* the transaction to nodes $\{v_1, \ldots, v_{15}\}$

In practice, prior to nodes undertaking the *transaction balloting* protocol, the network must maintain an understanding of the transaction *gossiped* by node v_1 in such a way that as the network experiences *node churn*, the transaction information will not be lost. Figure 2 reflects a network state, post the *gossiping* of the transaction information, where nodes $\{v_1, v_2, v_3, v_4, v_7, v_8, v_9, v_{15}\}$ have exited the network and where nodes $\{v_5, v_6\}$ are *dishonest* – i.e., many nodes with an understanding of the transaction have effectively lost the transaction information and, therefore, cannot propagate that information at the next *epoch* during the *transaction balloting* protocol. However, the transaction information is not lost and will still be propagated.

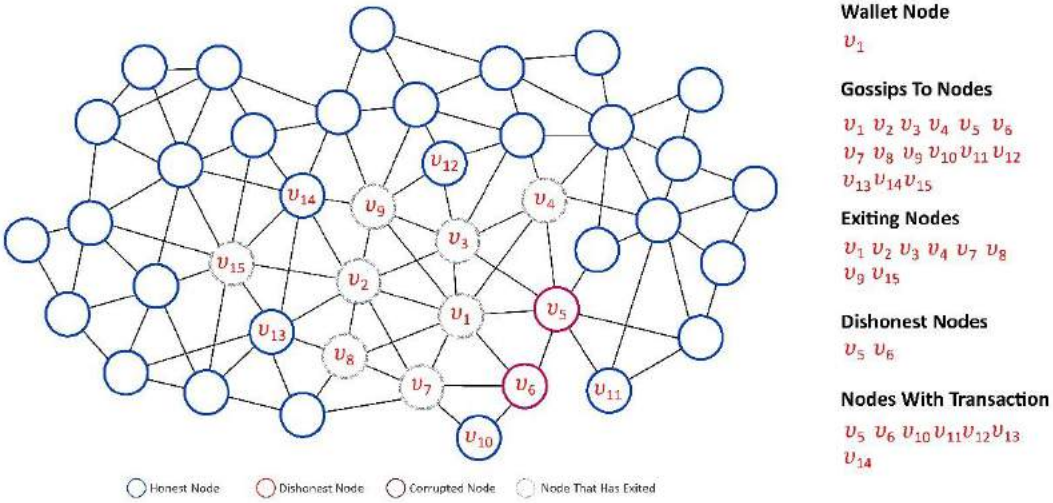

Figure 2. Nodes $\{v_5, v_6\}$ are *dishonest*, while others have exited, but the transaction persists

By requiring that the d highest aged peers, where d is set to 15, be pushed the transaction being propagated, the probability of the transaction reaching 32,758 nodes in the network approaches 1. At the time of this writing, the largest public blockchain averages approximately 10,000 active nodes – i.e., by setting d to 15, the probability of saturation would approach 1,

even though it is not required by the Proof of Majority consensus mode. To manage smaller or larger networks more efficiently, the value of d can be adjusted without causing *hard forks* in the blockchain. Whatever the value of d, the goal is to propagate the transaction to as many nodes as necessary to mitigate transactions loss due to *high churn rates* and to mitigate certain attack vectors, particularly that of *dishonest* nodes colluding to control a majority of the network.

In the case, that two *honest* nodes collide, choosing each other to exchange transaction information with, the node that receives the message first, ceases outbound information exchange during the *transaction propagation* protocol.

7.3 Transaction Balloting

Forming a network wide consensus about the state of the network first means forming statements that are to be considered for accuracy by the rest of the network. These statements take the form of blocks composed of unconfirmed transactions. Nodes in the network receive unconfirmed transactions through the *gossip* protocols, but all nodes are not guaranteed to have received all unconfirmed transactions. Further, even if all nodes were guaranteed to receive all unconfirmed transactions through the *gossip* protocol, nodes would still need to agree on which transactions possibly represent the current state of the network and then broadcast possible states, as blocks, for consideration by the network for inclusion in the blockchain. Therefore, a protocol for forming this set of unconfirmed transactions is required – i.e., a protocol for forming a ballot of unconfirmed transactions that permeates nodes, network wide.

Transaction balloting is essentially information synchronization of all unconfirmed transactions previously *gossiped* to network nodes that meet certain criteria. All unconfirmed validated transactions are handled by all nodes; no discrimination is allowed. A network wide ballot is constructed through peer-to-peer communications, sized to allow the underlying unconfirmed validated transactions to all fit into the next block being created. Unconfirmed validated transactions outside the space limitations of the block are stored by the nodes, in their *unconfirmedtransactions* queues, for inclusion in the next block. Unconfirmed transactions are validated and ordered by time stamp for inclusion in each ballot submission endorsed by each node and propagated to node peers.

The primary task of each node, at this stage of an *epoch*, is to arrive at an ordered universal set of unconfirmed transactions based on the transaction ballots endorsed by its peer nodes. An iterative process of peer based universal set building is employed. Nodes propagate ballots to connected peers, starting with a ballot derived from their then current understanding of the universe of unconfirmed validated transactions. Nodes receive ballots from connected peers and add to their universal set those transactions on these ballots that are not on its own ballot. The ballot is ordered at each universal set augmentation. The new universal set ballot is then propagated again and again and the node updates its universal set ballot from the ballots received from connected peers. Nodes continue this iterative process for a determined set of iterations, known as a cycle. At the conclusion of the cycle, the node is satisfied that it has an ordered set of all unconfirmed validated transactions synchronized with the rest of the network. Characterized more formally:

Definition (token). A *token* is the *hash* of a known unconfirmed validated transaction at any given node that predates the beginning of the current *epoch*.

Definition (node ballot). A *node ballot* is the ordered universal set ξ of all *tokens* known to a node and those *tokens* subsequently added via connected node endorsement from a set of connected nodes V such that $\xi \rightarrow \xi \cup 2^{V}$.

Definition (lower bound). A *lower bound* on the number of iterations needed for the deterministic data forwarding algorithm to disseminate a *token* in a dynamic network topography, exists and be calculated for the set of connected nodes N in the network as $O(\log N)$ [8][7].

Definition (cycle). A *cycle* is the number of iterations of internode *token* dissemination equal to the *lower bound*, in which each node selects from its *randompeer* queue, the *active* peer with the *highest* age, that has not yet been selected during the *cycle*, to affect a bidirectional exchange of *node ballots* where each node derives a localized ordered universal set of unconfirmed validated transactions – i.e., where the localized universal set U is derived such that $\forall v \subseteq U \; and \; U \supseteq \forall v$.

Because the *lower bound* may be larger than the size *randompeers* queue, due to the *peer sampling* protocol being unable to secure the requisite number of random peers or that there are not enough *active* peers at the time of processing, it is possible for the node to exhaust the *randompeers* queue, using the method of selecting the previously unselected *highest* aged *active* peer, before the *cycle* concludes. If this occurs, the node will select random entries from *randompeers* queue to manage the remaining iterations in the *cycle*.

Since block creation occurs only once per *epoch*, transaction balloting is assumed to be executed only once per *epoch*, by each node n, using the following schemas centered on a push/pull approach:

> Once per *epoch, n*ode n performs a *cycle* of *node ballot* exchanges with connected peers p derived from the *randompeers* queue, such that the number of exchanges is equal to the *lower bound*:
>
> 01. If a new *node ballot* for the *epoch* does not exist, then create a *node* ballot from known *tokens*
> 02. Select p as the highest aged descriptor in the *randompeers* queue, that has not been previously selected in this *cycle*, that represents a *active* peer, and that is not in the *suspectednode* queue
> 03. Package a buffer b with the *node ballot*
> 04. Send b to p and set answer flag f to 0 for p
>
> Node n is receiving a buffer b_1 from another node p:
>
> 01. If a new *node ballot* for the *epoch* does not exist, then create a *node ballot from known* tokens
> 02. If p is not in the *suspectednode* queue, then

a. If f is set to 1 for p, then
    i. Package a buffer b_2 with the *node ballot*
    ii. Send b_2 to p
b. Update the *node ballot*, as follows:
    i. For each *hash* h in b_1 process as follows:
        1. If h is not contained in *node ballot*, then retrieve transaction t represented by h from p
        2. If t cannot be retrieved or t is invalid, then add p to the *suspectednode* queue
    ii. If p is not in the *suspectednode* queue, then
        1. Append b_1 to the *node ballot*
        2. Remove duplicates from the *node ballot*
        3. Order the *node ballot*

03. Set answer flag f to 1 for p

At the conclusion of the cycle, the probability approaches 1 that there exists a single ballot of transactions network wide. Any nodes that have a transaction ballot, that is not synchronized with the network, will start the block building protocol with an invalid ballot – i.e., the node's ballot will not equal ξ. Thus, the node will create an invalid block that will be ultimately not be accepted during the majority consensus protocol phase of network wide processing. As an inverse to probability approaching 1 that there is a single network wide transaction ballot (ξ), the probability of a node having a transaction ballot different that is different than ξ approaches zero. Therefore, the effects of the resultant invalid blocks on the network wide consensus will, also, approach zero.

Consider the network in Figure 3 that reflects the network state, just after the first round of internode information exchanges during the *transaction balloting* protocol, assuming the starting state was of the network was that attained post propagation of transaction information from the wallet node v_1, where nodes $\{v_1, v_2, v_3, v_4, v_7, v_8, v_9, v_{15}\}$ have exited the network and where nodes $\{v_5, v_6\}$ are *dishonest* and, therefore, it can be expected that they won't accurately participate in the *transaction balloting* protocol.

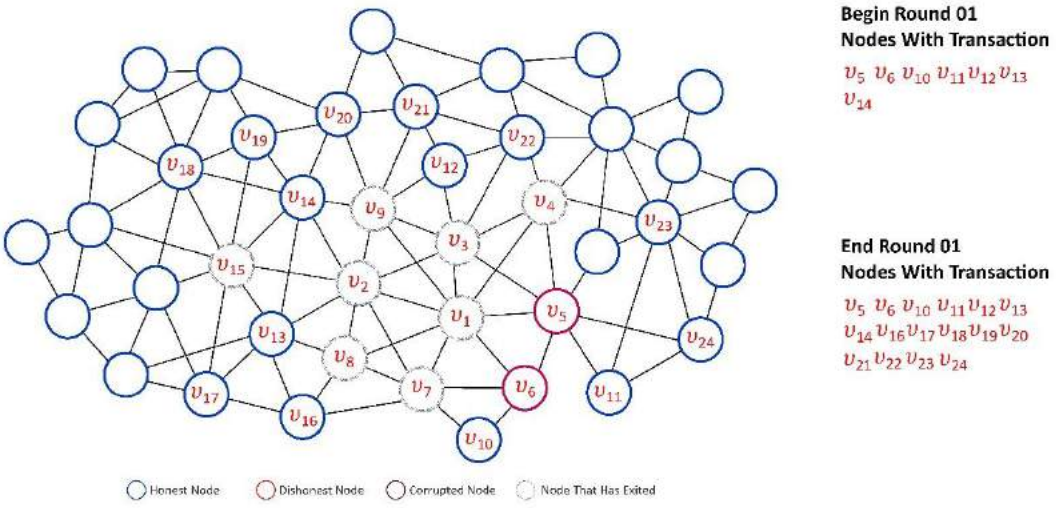

Figure 3. First round of information exchanges during *transaction balloting*

Given that the number of internode information exchanges, the entire network produces, is equal or greater to the *lower bound*, the network will converge. In figure 4 that after the second round of information exchanges, virtually all nodes have an accurate understanding of the *transaction ballot*.

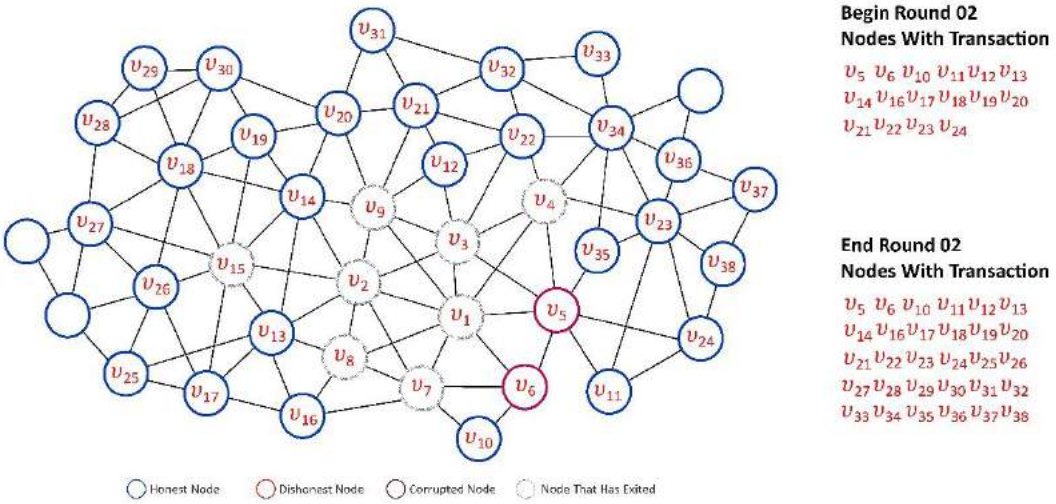

Figure 4. Second round of information exchanges during *transaction balloting*

By the conclusion of the third round of information exchanges, figure 5 shows that the entire network has converged – i.e., all nodes in the network now has an accurate understanding of the *transaction ballot* and are ready for block creation and *network wide voting* protocol.

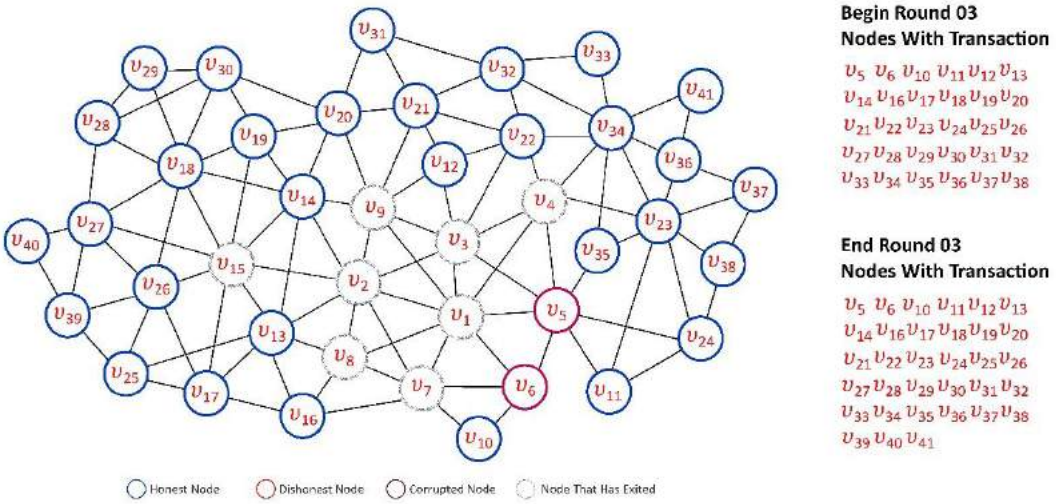

Figure 5. Third round of information exchanges during *transaction balloting*

In the case, that two *honest* nodes collide, choosing each other to exchange transaction information with, the node that receives the message first, ceases outbound information exchange during the *transaction balloting* protocol.

7.4 Network Wide Voting

In most consensus protocols, nodes exchange messages asserting statements about the state of the network to other nodes. Leveraging public keys as the basis for naming conventions used for nodes and where nodes are required to digitally sign internode messages, we assume that such statement assertions cannot be forged. In the Proof of Majority consensus model, when a node receives a set of statement assertions from other nodes that can form a majority (50%+1), it assumes no properly functioning node ("an *honest* node") will ever contradict that statement. To permit progress in an environment of node failures and/or *dishonest* nodes, a node is permitted to use its own assertion as part of the majority when arriving at a majority decision. Characterized more formally:

Definition (majority consensus). A *majority consensus* is a pair $\langle V, M \rangle$ comprising a set of nodes V and a majority function $M : V \rightarrow 2^{2^V} \setminus \{\emptyset\}$ specifying one or more majority sets for each node, where node belongs to all of its own majority sets and acts as a tie breaker – i.e., $\forall v \in V,\ \forall q \in M(v),\ v \in q$.

Definition (majority). A set of nodes $U \subseteq V$ in the majority consensus $\langle V, M \rangle$ is a *majority* $iff\ U \neq \emptyset$ and U contains a majority set for each member – i.e., $\forall v \in U,\ \exists q \in M(v)$ such that $q \subseteq U$.

A majority consensus is achieved when a set of nodes, in the network of nodes, has been convinced that one specific statement, being asserted by the network from among the set of statements being asserted, is the correct statement, and that set of nodes convinced of that one specific statement's correctness is the majority of the nodes in the network. Traditionally, a Byzantine consensus of this type requires that all nodes in a network accept the same statement as correct, therefore requiring that all nodes in the network be known and can be polled to determine which statement each node accepted as correct. Polling must be done by a centralized authority, in this case, and all nodes must be known prior to polling. This precludes open membership in the network and it precludes decentralized control. To allow for open membership and decentralized control, Proof of Majority leverages the replication algorithm commonly known as Practical Byzantine Fault Tolerance ("PBFT") [10], wherein the replication system is typically composed of $3f + 1$ nodes, any $2f + 1$ of which asserting the same statement as correct, constitutes a majority; where f is the maximum number of Byzantine failures the system can survive before corruption occurs. Byzantine failures in Proof of Majority are nodes in the network acting intentionally or unintentionally to corrupt the network wide accepted statement (known as "*dishonest* nodes"). The fundamental innovation, beyond the PBFT, of the majority consensus model, is that each node v is allowed to choose which statement it believes is correct from the set of statements $M.v/$ being asserted by the network. Thus, network wide majorities are arrived at from individual decisions made by each node. It is assumed in the majority consensus model that no individual node has a complete knowledge of all the nodes in the network, unlike the requirement in the traditional PBFT system, yet network wide consensus is still possible.

Consider the eight-node peer group in Figure 6, where each node is asserting a statement for consideration of correctness. Lines represent peer-to-peer connections in the peer group, through which messages reflecting a given node's assertion is transmitted to other nodes. *Honest* nodes $\{v_1, v_2, v_3, v_4, v_7, v_8\}$ are asserting a correct statement about the network state,

while *dishonest* nodes $\{v_5, v_6\}$ are not. Independently, the *honest* nodes will still arrive at a correct statement regarding the network state. Thus, the majority of the network will reflect an accurate statement of the network state, without each node arriving at a decision of accuracy and then having to transmit that decision to the other nodes.

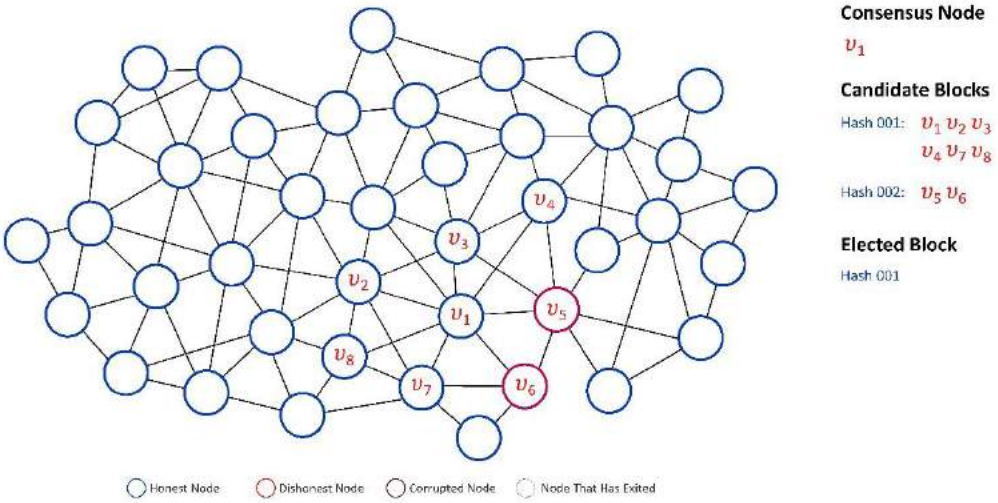

Figure 6. Node v_1 *a*rrives a majority consensus despite *dishonest* nodes $\{v_5, v_6\}$

In practice, the statement about the state of the network is reflected in the block being created during any given *epoch*. Nodes create blocks once an *epoch* from the *transaction ballot* arrived at using the *transaction balloting* protocol during the given *epoch*. The *hash* of the created block is then exchanged only with connected peers – i.e., there is no *gossiping* of the block nor an exchange of the block preceding the vote. Each node determines the majority consensus of the blocks created by its peers using only the *hash*. Once the node has arrived at majority consensus, the winning *hash* is compared to the *hash* of the block it created. If it matches then it has the winning block. If it does not match, then it asks the first node that it received the winning *hash* from for its block. Once validated, that block is now the node's new block and is added to its blockchain. If the received block is not valid, then it asks the second node that it received the winning *hash* from for its block and so on, until it receives a valid block. If it never receives a valid block, it keeps its block and a potential fork occurs that will eventually be abandoned.

A majority consensus is achieved network wide at each *epoch*. Each node in the network will maintain entries in a *hashvotes* queue, such that the queue contains one and only one *hash* for a connected node representing the block that node created during the given epoch. The *hashvotes* queue also includes a descriptor for the block creator and a timestamp. Each node in the network, determines a majority consensus once during a given *epoch*, using the following schemas centered on a push approach:

> Node n performs a single block *hash* h unidirectional exchange with each of its connected peers p:
>
> 01. Add the h for its own block to the *hashvotes* queue

02. Select p as complete list of the connected *active* peers for n that are not in the *suspectednode* queue and not in the *timeblocked* queue
03. Package a buffer b with h
04. Send b to p
05. Set processing block flag f_1 to 0
06. Set the retrieving block flag f_2 to 0

Node n is receiving a buffer b containing hash h from another node p:

01. If p is not in the *suspectednode* queue and not in the *timeblocked* queue, then
    a. If p is not in the *hashvotes* queue, then add p and h to the *hashvotes* queue
    b. If *hashvotes* queue is complete, then
        i. Tally the number of different *hash* values contained in the *hashvotes* queue
        ii. If there is a tie, tally again skipping the *hash* value entry for node n
        iii. If there is a tie, select the entry for the tied *hash* value that was received first
        iv. If the winning *hash* value is equal the *hash* value for the block in node n, then node n has a valid block to be processed:
            1. Add the first node descriptor of the winning set of *hash* values to the *timeblocked* queue
            2. Set processing block flag f_1 to 1
02. If the processing block flag f_1 is set to 0, then set the retrieving block flag f_2 to 1

Node n has a retrieving block flag f_2 set to 1:

01. Iterate through the winning peers w, until a block is received:
    a. If the winning peers w has been exhausted without a valid block, then
        i. Set processing block flag f_1 to 1
        ii. Set the retrieving block flag f_2 to 0
    b. If the block has been received and it is not valid, then
        i. Add the node w descriptor to the *suspectednode* queue
        ii. Set processing block flag f_1 to 1
        iii. Set the retrieving block flag f_2 to 0
    c. If the block has been received and it is valid, then
        i. Replace the block from node n with the retrieved block
        ii. Add the first node descriptor of the winning set of *hash* values to the *timeblocked* queue
        iii. Set processing block flag f_1 to 1
        iv. Set the retrieving block flag f_2 to 0
    d. Package a buffer b with a block retrieval request
    e. Send b to w

Node n has a processing block flag f_1 set to 1:

01. Process the nodes block, as follows:
    a. Establish a new *timestamp* for the block
    b. Calculate a new *hash* for the block
    c. Add the block to the blockchain

The key to the *network wide voting* protocol is that all nodes start with the same *transaction ballot*. Assuming that the *transaction ballot* protocol has created a convergent network during the current *epoch*, then each node participating in the *epoch* will reach the same conclusion in this deterministic network. Because the network has been shown earlier in this paper to reach a convergence with a probability only approaching 1 but not guaranteed to be 1, it can assumed that there will exists nodes on the network with a *transaction ballot* that differs from the rest of the network. These nodes will deterministically create blocks that though accurately represent the transactions in their *transaction ballot*, don't participate in the consensus that will be arrived at by the majority of nodes. These nodes are not *dishonest* and based on the schemas above, these nodes will still add the block *elected* by the rest of the network to their blockchains, assuming that a majority of their connected peers are *honest* and have started with the complete *transaction ballot*. If the majority of their connect peers are *dishonest*, a fork will occur that will eventually be abandoned.

Winning *hashes* are not *gossiped* among nodes, primarily to prevent certain attack vectors, particularly that of *dishonest* nodes flooding the network with invalid *hashes*. As a result, nodes are left to *elect* blocks for the blockchain based on a majority voting consensus rather than an externally disseminated vote that can't otherwise be validated in a decentralized network.

Proof of Majority is at its core a probabilistic consensus, where each *epoch* introduces more certainty over previous blocks, eventually reaching enough certainty that the likelihood of a different history being adopted by the network is sufficiently small. A block is *committed* if the majority of the nodes add their weight onto the blockchain that the block belongs to. Although it has been shown earlier in this paper that this model will produce a consensus and a block will be *elected* to the blockchain in a fault-tolerant manner, the questions of safety and liveness are still open.

The Fischer Lynch Paterson impossibility result [11] states that a deterministic asynchronous consensus system can have at most two of the following three properties:

01. Safety; results are valid and identical at all nodes
02. Liveness or guaranteed termination; nodes that don't fail always produce a result
03. Fault tolerance; the system can survive the failure of one node at any point

This is a proven result. Any distributed consensus system must sacrifice one of these properties in order to have perfection of the two properties, if it operates in a deterministic manner and is fully asynchronous. However, there are varying degrees of each, that can be attributed to a specific system. The realistic goal, then, is to achieve near perfection, but not perfection, in each of these areas.

It has been proven that to tolerate f *dishonest* nodes, you need $2f + 1$ *honest* nodes in a given network [10][12]. It is not possible for[an asynchronous system to provide both safety

and liveness with more than this number of *dishonest* nodes. As a result, ensuring that the network has $2f + 1$ is the first step – i.e., at any point in time within the network, there must be more *honest* nodes, that will remain honest sufficiently long, than there are *dishonest* nodes. Assuming the network can achieve and maintain this ratio, complete tolerance to fault can be approached. Because of the same protocols that allow the network to approach near complete tolerance to fault, information sharing among nodes will also be complete with a probability approaching 1. Informally speaking, the protocols introduced in this paper favor safety and fault tolerance over liveness. However, liveness is not abandoned in this model. Through the use of *epochs* nodes have a defined *window* of time, in which to complete their work or move on to new work in a new *epoch*. Assuming nodes are not deadlocked internally, nodes are effectively guaranteed to produce a result or terminate processing – i.e., node interactions won't cause deadlocks, only failures internal to the node cause a node to fail, thus nodes should receive *epoch* related network identifiers through regular internode messages unless they have failed internally, and, as a result, will complete their processing and move on to processing related to the new *epoch*. Internode communications are always bound by a timeout, where communications from the timed-out node are ignored until the next *epoch.* Either a new *epoch* or a timeout will prevent a pending internode deadlock, thus virtually ensuring liveness in the network.

In the unlikely event that there is tie among winning block hash values, the nodes will select the first winning block ordered by hash value. This situation can occur if the network does not converge during the *transaction balloting* protocol and may result in some nodes generating an alternative understanding of the blockchain – i.e., a fork that the network will automatically handle.

7.5 Forks and Abandonment

In Figure 7, below, the *honest* nodes will arrive at and maintain a correct statement about the state of the network. In other words, they will have added a correct block to the blockchain. It is assumed that the *dishonest* nodes will remain *dishonest* and will thus maintain a blockchain where the last block is now inaccurate or corrupt.

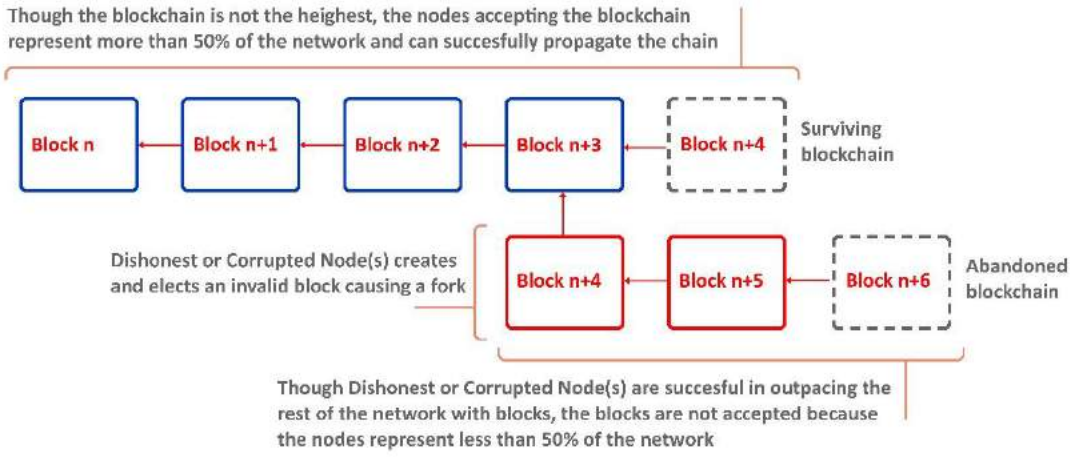

Figure 7. A fork has been created by *dishonest* or *corrupted* nodes

This represents a fork of the blockchain or a portion of the network that is corrupt. It is further assumed, that *dishonest* nodes will continue to assert false statements about the network state – i.e., build corrupt blocks and assert their correctness to their peer nodes. To prevent network wide corruption or the adoption of incorrect statements about the network state, these forks must be abandoned quickly.

The majority consensus model introduced in the paper, will quickly and automatically orphan forks created by *dishonest* or *corrupted* nodes. Because any block that, though valid, does not conform to the majority of valid inbound blocks from connected active peers will be ignored by nodes as they work to *elect* a new block onto the blockchain, a *dishonest* node can only succeed in *corrupting* nodes that *elect* its block. To *elect* its block would require that the majority of the node's connected active peers also send it the same block. This would require that a majority of the node's connected active peers also be *dishonest* or *corrupted*. At the next *epoch* the now *corrupted* node would then send a block, based on a *corrupted* blockchain, to its connected active peers, who would in turn ignore the block because it is not among the majority. The fork would be effectively orphaned. Therefore, a *dishonest* node would have to collude with a number of other *dishonest* nodes sufficient to overrun the network – i.e., *dishonest* nodes would have to represent more than half of the network. Processing and communication protocols in this consensus model prevent this.

A node that has become *corrupted* is not removed from the network. Instead, it will be placed on *suspectednode* queues in those nodes it is connected to and will be ignored for a certain block height. Eventually, the *corrupted* node will be given a chance to re-enter active communications with its connected active peers. As it re-enters active communications, it will determine that its understanding of the blockchain differs from the network wide understanding and it will seek to synchronize. Once synchronized, it can again participate in *epochs*. Thus, attempts to *corrupt honest* nodes by *dishonest* nodes has only a short-term impact on the network.

Because each node in the network at each *epoch* converges on the same set of transactions to process through the *transaction propagation* protocol, each node will produce the same block for propagation during the *network wide voting* protocol. The only variance will be *dishonest* or *corrupted* nodes. Thus, devoid of *dishonest* nodes, the network should not produce forks, unless convergence is not achieved during the *transaction balloting* protocol.

7.6 Pacing Votes

Nodes are allowed to propagate blocks for peer voting as long as they are not on a peer node's *timeblocked* queue. Each node in the network tracks the first peer node that delivered a block that was *elected* onto the blockchain. That first connected node is placed into the *timeblocked* queue to prevent that connected node from sending new blocks for voting for a calculated number of *elected* blocks. Because *dishonest* nodes can act like *honest* nodes for a period of time before embarking on *dishonest* behaviors, Proof of Majority requires *honest* nodes that act quickly to *abstain* from block voting for a determinant period of growth in the blockchain. Nodes must be responsive. Based on the processing requirements this consensus model places on nodes, most nodes will be very responsive with timing footprints driven more by internode messaging timeframes than computing timeframes. As a result, it is safe to assume that any node, consistently *electing* blocks at a more rapid rate than nodes in its random peer

collective, is acting out-of-band – i.e., pacing that node's block voting speed is thus prudent. Any pacing requirement must simply be large enough to mitigate a *dishonest* node from altering a consecutive block history. For this consensus model, an increase in blockchain height of 1 is a sufficient blockchain addition before allowing a node to *elect* another block onto the blockchain.

Assuming that each node in the network is achieving the requisite number of connected active peers, the *vote pacing* protocol will result at most in 1/32 of the network being on a *timelocked* queue. In practice, 1/32 of network being in a *timelocked* queue is highly unlikely, as it would require that network wide the *timelocked* queues collectively contain only one instance of any given node. Because nodes, by design, will be members of many random peer collectives, thereby creating an unpartitioned network, this is highly unlikely to occur. In this paper, this probability of this is left to the reader, since 1/32 of the entire network *abstaining* from block voting at any given *epoch* is an acceptable trade off while mitigating attack vectors.

7.7 Trusted Peers

Trust related information is not broadcast to connected peers nor is it *gossiped* to the network at large. Nodes collect and maintain trust information on connected peers only and use heuristics domiciled at its node to collect that trust information about nodes requesting connection and about nodes its operates with post initial connection. This is done primarily to prevent *dishonest* nodes from passing on negative trust related information about *honest* nodes, thereby negatively impacting the network at large by causing network nodes to effectively suspend communications with *honest* connected active peers.

Breaking node protocols by attempting to exchange information out-of-band, attempting to communicate too often, attempting to communicate too quickly or too slowly, communicating with inaccurate or inactive encryption keys, are all reasons why a node will place a connected active node on its internal *suspectednode* queue – i.e., if a node is not strictly obeying the rules of the then active protocol during any given *epoch* it will effectively be suspended and ignored by the connected node.

7.8 Peer Aging

Before a new node is allowed to submit blocks for a network vote and before it is allowed to exchange any network state related information, the node must *age*. Aging is marked by the number of blocks added to the blockchain since the node initiated a peer request with another node. Aging is tracked by the node receiving the peer connect request. Aging information is not broadcast to peer nodes nor is it *gossiped* to the network at large. Peer aging, along with *suspectednode* and *timelocked* queues are critical components of mitigating attack vectors. These are the primary mechanisms ensuring that *dishonest* nodes must work significantly harder than the rest of the network combined to achieve majority and propagate an alternative history of the blockchain. In combination with the other protocols described in this paper, this consensus model is hardened against attack vectors. By requiring an aging event to occur prior to accepting information exchange, the network effectively and efficiently slows new entrants from disseminating inaccurate, malicious, or otherwise malformed information that would otherwise unduly burden *honest* nodes, and thus compromise its *honest* activities.

Aging is calculated as a factor of blockchain height growth, where the primary goal is to slow node entry into the network under the assumption that the all nodes are just as likely to be *dishonest* and they are to be *honest*. The focal activity, among the many potentially *dishonest* activities, prevented by aging is a *dishonest* node attempting to overburden *honest* nodes with malicious exchanges of information – i.e., denial of service or spamming. As a result, any aging requirement must simply be large enough to mitigate a loss of processing power by an *honest* node due to malicious information exchanges initiated by a *dishonest* node. For this consensus model, an increase in blockchain height of 10 is sufficient.

7.9 Node Synchronization

Proof of Majority requires any node, whose understanding of current *epoch* is different from that being disseminated during the p*eer sampling* protocol, to synchronize its blockchain with the rest of the network. Having a different understanding of the state of the network can happen in several ways:

01. **Node Churn**

    The network is not static, instead nodes are continuously entering and exiting the network, a phenomenon commonly called *churn*. When a new node enters the network, it enters per the *node discovery* protocol. Once aged, the connected active peers provide the new node with the next *epoch* identifier. Entering nodes are never allowed to participate in the current *epoch*. This is necessary to make sure that each *epoch* converges to the average that existed at the start of the *epoch*. Continuously adding new nodes during an *epoch* would decrease the ability of the network to converge. There is no explicit mechanism for a node to exit the network. Instead, timeouts are used to terminate abandoned information exchanges. Terminated information exchanges, due to real or presumed node failures, do not adversely affect the network's ability to converge at any given *epoch*. Thus the network is self-healing – i.e., it removes failed nodes from the system automatically.

02. **Suspectednode Queue**

    Nodes that are not behaving per the *trusted peers* protocol are placed on the internal *suspectednode* queue of any connected active peer that observes related infractions. This information is not forwarded to any other connected active peers (including the node placed on the *suspectednode* queue) nor is it *gossiped* to the rest of the network. This prevents *dishonest* nodes from disrupting the network by simply forwarding or *gossiping* erroneous trust information to other nodes in the network. A node will also be placed onto a *suspectednode* queue if it either timeouts during an information exchange or if it delivers a block during the *network wide voting* protocol that does not confirm to the majority of blocks up for *election*. Because this consensus model has no means to determine if a node warranting placement on a *suspectednode* queue is *dishonest* or has been *corrupted* by a *dishonest* node, placement on a *suspectednode* queue does not mean permanent exile. Instead, a node on a *suspectednode* queue is ignored until the blockchain height grows sufficiently to obscure any malicious or erroneous information the node is or was attempting to exchange – i.e., all communication attempts while on a

suspectednode queue are ignored for a determinant number of blocks, not permanently. For this consensus model, an increase in blockchain height of 250 is a sufficient blockchain addition before allowing a node to be removed from a *suspectednode* queue. Because there is no distinction between a *dishonest* node and a *corrupted* node, this consensus model allows for all nodes to eventually attempt to reenter in the network and to follow the rules of all the network protocols – i.e., keep the network as large as possible, even if there is a constant mitigation of *dishonest* nodes.

03. **Timelocked Queue**

Nodes that are behaving per the *trusted peers* protocol will periodically be placed on the internal *timelocked* queue of any connected active peer in which the node was the first to deliver a block *elected* onto the blockchain during the *network wide voting* protocol. This is in adherence to the *pacing votes* protocol and helps to ensure that *dishonest* nodes temporarily acting *honestly*, don't take advantage of their *trusted peer* status to attempt to alter the network understanding of the blockchain to one that is erroneous.

Unlike nodes on the *suspectednode* queue, nodes on the *timelocked* queue are not ignored to prevent that node from being forced to reach a timeout state with one or more of its connected active peers during the *epochs* before it is removed from a *timelocked* queue. Although, the *timelocked* node is not allowed to exchange information, it will receive response messages *reminding* it that it is on a *timelocked* queue – i.e., while nodes are acting *honestly* they should not be penalized, certainly not in a manner that will increase overall network latency.

In all cases, before peer nodes will respond to a node requesting blockchain information, or any other information set, the requesting node must first meet the requirements of the *peer aging* protocol.

In the presence of majority posterior corruption, it is impossible to achieve consensus on the correct historical blockchain at any given *epoch* without an additional trust assumption – i.e., a centralized repository of the blockchain. Since it is undesirable to add this type of trust assumption into the Proof of Majority consensus model, another mechanism must be employed wherein the network can converge, even if the node requesting a blockchain history cannot. In the case where the node cannot achieve convergence with the rest of the network, the node's blockchain will represent a fork from the network blockchain and the node will quickly find itself on one or more *suspectednode* queues; thereby effectively being eliminated from information exchanges with connected peers until such time as it can converge with the rest of the network relative to the network wide understanding of the correct blockchain.

To retrieve the current blockchain, the requesting node sends a request message to each of its connected active peers. The node will receive chains of hashes from its connected peers representing the blockchain that is understood by the connected peer. The node will then verify the chain and reject it if it is found to be invalid. If a received chain is valid but deviates from a node's current chain too far in the past, such a chain is not punctual and will be rejected – i.e., if the chain deviates 40 blocks prior to the current block height. If a received chain is valid but deviates from a node's current chain too far in the future, such a chain will be

rejected – i.e., if the block height minus the height of the blockchain the node understood is more than a 20% deviation from the projected blockchain height based on the time elapsed between the then current time and the timestamp of the last block on the blockchain the node has a current understanding of. A node always chooses the longest chain among all the chains that it did not reject, by computing the longest valid chain that is a prefix to the majority of those chains and then by finding the longest chain in that list that contains the prefix chain. If this is the network start then the genesis block is the blockchain.

In this way, nodes joining or rejoining the network after a short or long period away from the network, can achieve convergence with the network regarding the correct understanding of the blockchain. If the node cannot, due to the presence of majority posterior corruption, the node will not corrupt the rest of the network and instead will find itself effectively isolated from information exchanges until such time as it can achieve convergence with the rest of the network – i.e., this is a self-healing decentralized network.

7.10 Node Startup

By design, the nodes in the network are interconnected to form a random graph. A node attempting to enter the network first queries a set of DNS servers run by volunteer operators, which return a random set of bootstrap nodes that are currently active in the network. The node then sends connection requests to these active nodes and waits to meet the requirements established by the *peer aging* protocol. Once the node has met these requirements, the node is allowed to query active nodes for the network's current understanding of the blockchain through the *node synchronization* protocol. It also establishes its first *randompeers* queue via the *peer sampling* protocol, from which it establishes the requisite number c of connected peers. To virtually ensure network convergence during *gossip* based protocols, c is set to 32. Each node attempts to keep a minimum number of connections c to other active nodes open at all times. Should the number of open connections drop below c, the node will randomly select an address from its *randompeers* queue and attempt to establish a connection. If incoming connection requests would otherwise exceed number of requisite open connections, the connections are not abandoned – i.e., node can have more than c connections, but strive to have at least c connections. A node's total number of open connections is therefore likely to be higher for nodes that also accept incoming connections – i.e., not all nodes accept incoming connections as they may be behind firewalls or network address translations.

Partitions in the network connection graph are not actively detected and if they occur the partitions will continue operating independently. This is certainly not desirable, from a liveness point of view, and will cause the network wide understanding of the current state of the blockchain to diverge, creating more than one parallel and possibly incompatible transaction histories. It is therefore of paramount importance that network partitions are detected. However, without an additional trust element, typically achieved through centralized oversight, this is not easily accomplished. Since this is highly undesirable, it is of equally paramount importance that each node works to maintain its requisite number of connected active peers.

There is no explicit way to leave the network. The addresses of nodes that leave the network may linger for a small period before the active nodes purge them from their known addresses

set and stop sending them in answer to *peer sampling* protocol requests. Message exchange timeouts are the primary method employed to purge nodes that have left the network.

7.11 Node Footprint

Nodes in the network are very light weight with an extremely low processing footprint and a very small memory footprint. Information exchanges are never much larger than the size of a single block and movement frequency is naturally throttled at each *epoch*. Consensus building during the *network wide voting* protocol requires block hashes to be exchanged, not blocks. Hardware running nodes can be a fraction of the processing power required to run nodes in consensus models that require heightened levels of difficulty for node selection, because simple and quick hashing is all that is required. Network energy is at its peek during the *transaction balloting* protocol when the network converges on the queue of unconfirmed transactions to be included in the next *elected* block; during other protocols, network energy remains very low. During this peek energy state, the network is simply *gossiping* transactions, which it has been found that 96% of the transactions will be smaller than 1kB [13]. Thus, the peek energy state of the network is substantially smaller than that of other consensus models.

Node energy states peek during blockchain synchronization, which under nominal network conditions occurs once during node startup. This consensus model makes a distinction between a *full* node and *thin* node only relative to the amount of information maintained at the node and that is directly related to the node's understanding of the blockchain – i.e., *full* nodes maintain the full blocks, while *thin* nodes maintain block and transaction header information. Both *full* nodes and *thin* nodes, however, participate in block creation and the *network wide voting* protocol, where *thin* nodes temporarily maintain full transaction information for any unconfirmed transactions *gossiped* during the *transaction balloting* protocol – i.e., once a block is *elected*, the *thin* nodes will have abandoned the full block and full transaction information set in favor of block headers and transaction headers. This means that the download and storage requirements of *thin* nodes scales linearly with the amount time since the network start. For validation of transactions and mitigation of certain attack vectors, there exists two different trust models between *full* nodes and *thin* nodes. *Full* nodes can make transaction verifications against the entire blockchain, while *thin* nodes must instead link transactions to a block in the blockchain, ensuring that the network accepted the transaction, but without any ability to directly verify any given transaction – i.e., *thin* nodes rely on the network's self-healing characteristics to mitigate attack vectors that they cannot directly address.

Thin nodes are connected to the rest of the network under the same protocols *full* nodes follow. Thus, they are connected to many active peers, instead of only one active *server* node, as with many other consensus models. This prevents certain attack vectors that plague *thin* nodes in other consensus models and helps to ensure that *thin* nodes fully participate in the network and are thus entitled to their share of the network's mining rewards.

Because Proof of Majority has such low processing and memory requirements, nodes can be run on handheld devices – i.e., smart phones have the necessary processing power, memory, and bandwidth to operate within the network in a manner in which they can earn mining rewards.

7.12 Network Epochs

Proof of Majority assumes that *epochs* proceed in lock step at all nodes in the network. In a large-scale distributed system, this assumption cannot typically be satisfied due to the unpredictability of message delays and the different drift rates of local clocks. Given an epoch j, let Tj be the time interval from when the first node starts participating in epoch j to when the last node starts participating in the same *epoch*. In a completely asynchronous network the length of this interval would increase without bound given the different drift rates of local clocks and the fact that a new node joining the network obtains the next epoch identifier from an existing node, incurring a message delay. To avoid this problem, when a node participating in epoch i receives an exchange message tagged with epoch identifier j such that $j > i$, the node must stop participating in epoch i and instead starts participating in epoch j. However, because the majority consensus model saves network energy by allowing each node to build its blockchain devoid of block gossiping, the node that stopped participating in epoch i would produce a blockchain of insufficient height. Therefore, nodes must complete the epoch i before moving to epoch j, in this scenario. Intuitively, we can assume then that the network, as a whole, will complete a specific *epoch* only when the slowest node in the network completes that specific *epoch*. This is an undesirable limitation on overall network throughput. The average pace of *epochs* must be more deterministic than that of the slowest node at a given *epoch*. A timeout on message exchange between a node and a given connected peer is all that is required to set an average pace for *epochs*. To avoid reducing the overall number of participating *honest* network nodes, that timeout must be larger than or equal to Tj. Since this consensus model leverages a push/pull communication schema, which propagates messages super-exponentially, and if we assume that each message arrives within a prescribed timeout during all communications, we can obtain a logarithmic bound on Tj for each *epoch* j. More importantly, typically many nodes will start the new *epoch* independently with a very small difference in time, so this bound can be expected to be sufficiently small, thus requiring an *epoch* length such that it is greater than or equal to Tj and can be used as the network wide messaging timeout.

The end of any given epoch is demarked by all nodes in the network completing the n*etwork wide voting* protocol, either by being eliminated from processing due to the timeout on message exchange, as just discussed, or by receiving block election messages from all non-timed out connected peers. Because this protocol requires answers or timeouts from all connected peers, it can be intuitively assumed that each node will complete the current *epoch* at virtually the same time within an extremely small differential of time. Because of this situation, this consensus model will not impose a predetermined timeframe for an *epoch* to complete in.

Each node should start the next *epoch* immediately, to ensure the timeliest *election* of the next block onto the blockchain. The *peer sampling* protocol is the first step in any *epoch* and each related message exchange contains the *epoch* number of the current *epoch* – i.e., the height of the blockchain. If a node receives a *peer sampling* protocol message containing an *epoch* number that does not match the height of the blockchain it has on record, then the node cannot participate in the current *epoch* and instead must synchronize its blockchain with the rest of the network. In doing so, the node should not attempt to synchronize its blockchain through message exchange with the connected peer that first reported the *epoch* number to it; the *node synchronization* protocol will be used in this case.

7.13 Network Traffic

While most blockchain systems use an announcement protocol to broadcast the availability of information, where peers in the system then decide whether or not to ask for the information, Proof of Majority uses a strict push exchange or a push/pull exchange, where peers are expected to accept the information, as long as that information is pushed within the guidelines of the then active protocol during a given *epoch*. Announcement protocols have been shown to significantly increase latency times with a blockchain system [13], counter to the goal of the Proof of Majority consensus model, which seeks to reduce overall system latency times. It has been shown that for information sets less than 1kB, the roundtrip delay present in the announcement protocol is considerable [13]. The information added to the system to announce availability of a new datum, to then respond with a request for that new datum, and then finally to send the new datum, is considerable in ratio to the datum itself – i.e., the smaller the datum the higher the ratio of overhead to information. Coupled with the three-fold increase in the number of internode exchanges to accomplish the datum transfer, there is cost savings to the system that can be represented, as follows, by simply sending datums directly:

Definition (announcement). An *announcement* message has a predefined nonvariant data size da that can be expressed in bytes. In a typical blockchain network, an announcement message is sent a peer node to inform the peer node that a transaction exists.

Definition (data request). A *data request* message has a predefined nonvariant data size dr that can be expressed in bytes. In a typical blockchain network, a data request message is sent back to the announcing peer node to request the transaction.

Definition (data send). A *data send* message has a variant data size ds that can be expressed in bytes and holds the transaction sent to a peer node.

Definition (savings). The cost savings cs to the network by sending the transaction directly to a peer node without an *announcement* message and without a *data request* message, can be represented as a percentage such that $cs \rightarrow (da + dr)/(da + dr + ds)$.

Given expected message sizes with in the Proof of Majority consensus model, the cost savings percentage can be expected to be 19.35% as follows:

$$da = 54\text{B}, dr = 54\text{B}, ds = 450\text{B}, cs \rightarrow (54 + 54)/(54 + 54 + 450) = 19.35\%$$

Because it has been found that 96% of all transactions on a representative blockchain system are smaller than 1kB [13], it is safe to assume that this savings percentage will be virtually constant throughout the life of the network.

The preceding reflects non-block information exchanges. The majority consensus model presented in this paper, does not require blocks to be *gossiped*, this prevents huge amounts of data from being propagated at each *epoch*. Instead, this consensus model is designed such that each node arrives at the next block on its own and waits for a peer related majority to emerge to *elect* that block onto the blockchain in a manner that ensures that the *honest* node

will have the same block network wide. If an *honest* node hashes a block that differs from the majority, then it will *ask* for the block from a connected active peer. During ongoing *epoch* protocols, where the network converges, nodes do not broadcast or *gossip* blocks. The cost savings on network traffic over the typical blockchain consensus model, therefore approaches 100%.

It has been shown that to achieve information saturation across the network, in a *push/pull* exchange model, requires $O(log\ log\ V)$ messages where V is the number of connected nodes. This assumes that complete saturation is required for the protocol in question. This consensus model only requires complete saturation during the *transaction balloting* protocol; the *network wide voting* protocol does not require complete saturation to achieve the necessary network convergence. The *transaction propagation* protocol is a *push* exchange model that has been shown to require $O(log\ V)$ message to achieve complete information saturation across the network, where V is the number of connected nodes. The *transaction propagation* protocol does not require complete saturation to achieve the necessary network convergence. Because the consensus model does not require complete saturation for all protocols during a given *epoch*, network traffic is bounded by the previous, but rarely meets that bound.

8. Blocks

Blocks on the blockchain represent a consensus among the nodes and reflect the network wide accepted statement about the state of the network at the time the block was added to the blockchain. The only mechanism for affecting change in the wallet is through the *election* of blocks onto the blockchain.

8.1 Block Creation

The majority consensus model presented in this paper, dictates that nodes create blocks devoid of a *coinbase* transaction and include all transactions received during the *transaction balloting* protocol. There are no fees built into any transactions and thus inclusion of any given transaction into a block is frictionless. Miners are not incented at the transaction level to mine, instead through the *voting rewards* protocol, miners receive compensation and are incented to act rationally and *honestly*. Blocks are also devoid of the typical *bits* field reflecting the current difficulty and are devoid of the typical *nonce* field – i.e., block creation is simply a matter of hashing the *balloted* transactions into a block while addressing the previous block's hash. Resource requirements for block creation are very small.

Nodes generate a single *candidate* block per *epoch* and propagate that block's hash, not the block, during the *network wide voting* protocol. Information exchange is therefore significantly reduced compared to traditional consensus models. Because each node will eventually select among the *candidate* blocks, it receives during the *network wide voting* protocol, for *election* onto its understanding of the blockchain, rather than receiving a block via broadcasting or *gossiping* from the network, the node must generate a time stamp for its *elected* block that will converge across the network. Since a time stamp will be generated after *election*, there is no need for nodes to include a time stamp during block creation. Therefore, the time stamp field in the block header will be set to 0 seconds from 1970-01-01 00:00. Once a block is *elected* and a convergent time stamp is set in the block header, the *electing* node will rehash the block and add it to its understanding of the blockchain.

When limits are imposed on block size, nodes will place the newest transactions, by transaction time stamp, from the *transaction balloting* protocol that won't fit into the block's prescribed size limit, back onto the *unconfirmedtransactions* queue for processing at the next *epoch*.

8.2 Block Time Stamping

Blocks are created and voted on within a connected peer group and although the network converges on the included transactions prior to block creation, blocks will not have time stamps that have converged. Thus, a mechanism that ensures all blocks across the various blockchain copies on the network, must be leveraged, if time stamps are to converge, as well. After block *election* during the *network wide voting* protocol, each node will establish the *elected* block's time stamp, such that; (1) the time stamp assigned to the block is the time stamp of the first transaction contained in the block, (2) if the time stamp of the first transaction in the block is at or prior to the time stamp *elected* in the last *epoch*, then the time stamp for the newly *elected* block is the difference between the time stamp from the block 5 *epochs* ago and the time stamp of the block 11 *epochs* ago, plus the median time stamp, in

seconds, of the last 11 blocks. In this way, this consensus model, ensures that block time stamps are ordered, that under nominal operating conditions, block time stamps represent the first transaction in the block, and that block time stamps converge across the network.

9. Voting Rewards

Transaction processing in a peer-to-peer network consumes resources and node operators within the network should be compensated for this consumption. However, this consensus model is frictionless, in that migrating a transaction from an unconfirmed state to a confirmed state is devoid of a fee. Node operators are compensated for participating in the *election* of new blocks to the blockchain by building blocks and submitting them to their connected peer nodes for voting. Blocks do not themselves contain any fee related transactions either. Instead, voting participation is tracked and participating node operators are periodically compensated through rewards to their wallet from revenues generated by the entire ecosystem.

Voting participation is reported on the blockchain through *heartbeat* transactions. Each node in the network tracks voting connected peers during the *network wide voting* protocol, by maintaining a *participatingnodes* queue of nodes that participated in a majority *elected* block. Each time a connected peer exchanges an *elected* hash with a node, that node increments a counter for the connect peer. This queue is not broadcast or otherwise *gossiped* to any other node. At each *epoch*, the node will uniformly randomly select a single connected peer from its internal *participatingnodes* queue, where the connected peer has a count of 2016 or more – i.e., the node must have more than intermittent participation during the period. The node will then place a transaction into the *transaction balloting* protocol that reflects the connect peer p and itself v. It will then reset the incremented counter for p in its *participatingnodes* queue – i.e., it will *gossip* a *heartbeat* transaction for the connect peer. As other nodes in the network receive the information during the *transaction balloting* protocol, they will validate the *heartbeat* transaction, such that; (1) p is not already in the *transaction balloting* queue, where duplicates are eliminated by keeping the first transaction ordered by the hash associated with v, (2) there is a *heartbeat* transaction for v in the last 2016 blocks of the blockchain just prior to the current *epoch*, and (3) there is no *heartbeat* transaction for p in the blockchain in the last 2016 blocks, just prior to the current *epoch*. If the blockchain is less than 2016 blocks in height, then requirement that there is a *heartbeat* transaction for v in the proceeding 2016 blocks is waived and the threshold for required vote participation is 1 block before a *heartbeat* transaction can be placed into the *transaction balloting* protocol.

A pro-rata distribution of a total network wide reward r is delivered periodically based on the nodes who have at least one *heartbeat* transaction on the blockchain – i.e., any given node with φ heartbeats, in a network of σ heartbeats, will receive a portion α of r for that period τ such that: $\alpha_\tau : (r_\tau \,/\, \sigma_\tau) * \varphi_\tau$. As long as the cost associated with mining is less than the reward for participation, rational nodes will continue to participate. Further, because nodes are dependent on their selection by a connected peer node to report *heartbeat* transactions on their behalf, and this selection is through a uniform random selection process, nodes are further incented to establish and maintain as many active peer connections as possible, increasing network interconnectivity and thereby reducing the probability of network partitioning and certain attack vectors. Additionally, the requirement that nodes sponsoring *heartbeat* transactions must also have had a node sponsor their *heartbeat* transaction prior, mitigates the risk of *dishonest* nodes simply placing themselves on the network and propagating *heartbeat* transactions for each other.

10. Energy and Latency

At the conclusion of the *epoch*, all network nodes have independently established the next block in the blockchain containing the same set of transactions. Because there is a probability approaching 1 that the blocks are created from the same base of transactions, we can assume that devoid of *dishonest* nodes, each node creates the same block. Further, we can assume that each node completed processing following the above schemas without the need to retrieve an actual block from a connected peer, with a probability approaching 1, as well. Therefore, network traffic was kept very low, while arriving at a majority consensus. Overall network energy is significantly reduced in comparison to typical blockchain implementations, while consensus is arrived significantly faster and is hardened against attack vectors.

11. Attack Vectors

Message *gossiping* is a major contributor to a network's vulnerability to most attack vectors. It has the potential to increase network traffic significantly thereby increasing network energy significantly. *Dishonest* nodes cannot easily *spam* a network where *honest* nodes do not *gossip*. This consensus model does depend on certain protocols exchanging information via *gossip*, so certain precautions have to be taken to mitigate attack vectors. To increase their chances of disrupting the network, *dishonest* nodes must establish more connections with the network by increasing peer-to-peer connections and/or by establishing more *dishonest* nodes. Because of the aging requirements, trust requirements, and internal queues that sequester bad behavior and slow even positive behavior, in this consensus model, *dishonest* nodes are materially slowed in their efforts to establish more peer-to-peer connections and to establish more *dishonest* nodes on the network. Overtime *dishonest* nodes could permeate a network, if aging were the only factor in determining trust. Therefore, acting like an *honest* node is the key to continued network inclusion. Once a node determines that a connected node is acting *dishonestly*, the *dishonest* node is ignored via placement on the *suspectednode* queue. There is a very low threshold for *dishonesty* in this consensus model. Even under nominal network conditions, devoid of *dishonest* nodes, it has already been shown that, as much as, 1/32 of the entire network node base will be *timelocked* though these nodes are behaving *honestly*. Thus, the network is constantly throttling node participation, without hindering overall network productivity, to mitigate attack vectors.

Aside from *dishonest* nodes attempting to overwhelm a network to inject illegal information, there are several attack vectors that a decentralized, democratized, network computer must defend against. Most attack vectors are successful because computing or information is not truly decentralized and not truly democratized. Traditional consensus models don't actually possess either of these qualities. Under most existing and operational consensus models, mining pools are massive and controlled by very few people. Though there are almost 10,000 nodes on the most popular blockchain today, almost no mining is done outside the 8-14 largest mining pools; this is not decentralization. Further, a single node wins the "lottery" or is chosen by some algorithm as the authority allowed to submit the next block for validation; this is not a democracy. Instead, these models create single points of failure and will always be open to some new attack. Further, the lack of traffic encryption between nodes further exacerbates the problem of mitigating attack vectors. Proof of Majority is truly decentralized, democratized, and all internode communications are encrypted – i.e., this consensus model can successfully defend against known attack vectors.

Mitigation against certain attack vectors is achieved through the use of node specific queues for *timelocking* honest nodes to pace their voting rates and placing nodes suspected of *dishonest* behavior on *suspectednode* queues, where all traffic from such nodes is ignored – i.e., nodes acting *honestly* are throttled assuming they will one day act *dishonestly*, while nodes acting *dishonestly* are immediately sequestered.

11.1 Double Spending

The problem of double spending was addressed in the original paper by Nakamoto, but only theoretically. There are several scenarios that plague typical consensus models, making them vulnerable to double spending attacks [14] at a very low cost to the attacker. Some can be mitigated or, at least, detected by, establishing a longer detection period; some cannot be mitigated this way. Information eclipsing is the primary door to successful double spending, especially in fast payment solutions [15]. The Proof of Majority consensus model is not open to the current double spending attack vectors, in that it does not present an opportunity for *dishonest* nodes to carry out the primary vectors:

01. **Race Attack**: Sending two conflicting transactions in rapid succession into the network. Because, at each *epoch*, this consensus model requires the network to converge on the queue of unconfirmed transactions to include in the next block, the network is not open to a single node being selected for block creation wherein the node arbitrarily selects the transactions it will include in the next block. Instead, all nodes create and come to consensus on the next block with the same total queue of unconfirmed transactions. Intuitively, conflicting transactions will be detected and found invalid, thus excluding the double spend of this type from the next block and blocks in the future.

02. **Finney Attack**: Pre-mine one transaction into a block and spend the same coins before releasing the block to invalidate that transaction. This vector requires a consensus model in which a single node is selected for block creation. That node can introduce a block onto the blockchain that then invalidates the original spend. This is not the case with the Proof of Majority consensus model – i.e., all nodes create the next block and come to consensus *electing* that block into the blockchain, so no single node has enough power to affect this type of double spend.

03. **51% Attack**: Usurp over 50% of the total computing power of the network to control which transactions appear in blocks including the ability to reverse transactions. Although this attack is possible, it has been shown that it is highly improbable in the Proof of Majority consensus model.

The double spend attack vectors that currently exist are not affective in this consensus model because they are dependent on a single node being selected for block creation during any given network *epoch*. This consensus model requires all nodes to participate in block creation and, thus, it is not open to double spend attack vectors.

11.2 Intentional Wallet Address Collision

It is possible that two different public keys will yield the same address. If such an address contains value, it would be possible for a *dishonest* node to redeem value from the associated

wallet. For the redemption to succeed, the *dishonest* node would need to find a private/public keypair such that the SHA3 256 of the public key would be equal to the Ripemd-160 preimage of 160-bit hash. Since SHA3 256 offers 128 bits of security, it's mathematically improbable for a single SHA3 256 collision to be found.

11.3 Sybil Attack

In decentralized network where inclusion in peer groups is open, a Sybil attack is possible. The possibility for a node to inject itself into the network and subvert a positive reputation by forging its identity to match that of a node that exited the network with a positive reputation, is an attack vector that must be mitigated. Encrypting information exchanges between connected peers helps to mitigate this risk, thereby requiring a *dishonest* node to somehow acquire a private encryption key held by the node it is impersonating. The push/pull information exchange protocols further mitigate this vector, by leveraging the fact that although it is easy to spoof an IP return address in an IP packet, it is not easy to subvert routing and receive messages on a spoofed IP address. A *dishonest* node is thus limited by the number of public IP addresses that it actually has control over. Proof of Majority also requires that peers use TLS connections with endpoints identified by IP address, thereby further reducing the risk of this vector's success.

11.4 Denial of Service and Spamming

The goal of the *dishonest* node or nodes can be to increase network energy to a point that the network slows and cannot efficiently, or at all, cycle through its protocols or *epochs*, thereby effectively crippling the network. This consensus model mitigates this vector by throttling network energy usage during all protocols. All nodes have a very low threshold for behavior that does not adhere to stated protocols and will sequester any node breaking this threshold. Nodes deemed to be misbehaving are immediately placed onto an internal *suspectednode* queue, where after they are completely ignored – i.e., any attempt to exchange information is rejected, thus they are unable to raise network energy. Any deviation from protocol is considered bad behavior; too many messages, out-of-band messages, sending invalid data, non-responsiveness, and the like, are all causes for sequestering at the first instance of the bad behavior. Using their *randompeers* queue, nodes initiated outbound communications with at most 15 random connected active peers during *gossip* based protocols during any given *epoch,* and during each *epoch*, nodes update their *randompeers* queue. So, at each *epoch*, nodes have new connected active peers to *gossip* with. Thus, nodes can quickly recognize out-of-band exchange attempts during any given protocol, resulting on the out-of-band node being placed onto the *suspectednode* queue – i.e., the nodes have many different ways to quickly determine if a node is misbehaving and then to subsequently ignore them.

11.5 Node Corruption

Dishonest nodes can slowly work to corrupt *honest* nodes by getting them to accept illegal or erroneous information and thereby corrupt the *honest* node relative to future *epoch* participation. Rather than overwhelming the entire network, *dishonest* nodes can simply overwhelm an *honest* node by establishing enough connections to represent a majority for the *honest* node. As such, the *honest* node can now be convinced that an illegal or erroneous

network state is accurate. Several of the consensus protocols are vulnerable to this type of attack vector and in a decentralized network where trust is distributed, the *honest* node will have no way to prevent this vector nor would it even be aware that it is under attack. As a result, the *honest* node will now undertake subsequent protocols in an *epoch* with illegal or erroneous information – i.e., the *honest* node is now acting *dishonestly* without intent.

For this to be an effective attack, the *corrupted* node, now acting *dishonestly*, must have means to spread the illegal or erroneous information or the erroneous information that succeeds it. To do this, it must connect with another *honest* node in the network. It's at this point that the *corrupted* node will be discovered by the *honest* node it attempted to exchange the erroneous information with. Because the connected peer is not overwhelmed by *dishonest* nodes, it will readily determine the misbehavior and place the *corrupted* node onto its *suspectednode* queue and ignore it – i.e., the *corrupted* node is quickly sequestered. At best, the *corrupted* node will create a fork in the network understanding of the blockchain; a fork that has already been shown will be quickly abandoned.

Nodes placed on a *suspectednode* queue are not eliminated from the network. Instead, they are temporarily sequestered from the network for a certain number of blocks. While throttling misbehavior to extremely low levels, levels that the network can easily absorb, it allows *corrupted* nodes to re-enter the network as well behaved *honest* nodes at a later time. Thus, *dishonest* nodes cannot slowly corrupt the entire network by systematically attacking *honest* nodes throughout the network over time, expecting the network to eliminate those nodes. Proof of Majority creates a self-healing network.

11.6 Honest Nodes Turning Dishonest

Nodes acting *honestly* for a period, gaining trust, and then acting *dishonestly*, is an attack vector that must be mitigated. In the majority consensus mode introduced in this paper, these nodes will be quickly categorized as *dishonest* and placed on a *suspectednode* queue and ignored. To reduce the attractiveness of this attack vectors, this consensus model paces *honest* nodes via the *pacing votes* protocol.

REFERENCES

[1] Daniel J. Bernstein, Niels Duif, Tanja Lange, Peter Schwabe, and Bo-Yin Yang. Highspeed high-security signatures. In *Cryptographic Hardware and Embedded Systems - CHES 2011 - 13th International Workshop, Nara, Japan, September 28 - October 1, 2011. Proceedings*, pages 124–142, 2011.

[2] Demers, A., Greene, D., Hauser, C., Irish, W., Larson, J., Shenker, S., Sturgis, H., Swinehart, D., Terry, D.: Epidemic algorithms for replicated database maintenance. In *Proceedings of the 6th Annual ACM Symposium on Principles of Distributed Computing (PODC'87), Vancouver, British Columbia, Canada, ACM Press* (August 1987) 1–12

[3] Karp, R., Schindelhauer, C., Shenker, S., Vöcking, B.: Randomized rumor spreading. In *Proceedings of the 41st Annual Symposium on Foundations of Computer Science (FOCS'00), Washington, DC, USA, IEEE Computer Society* (2000) 565– 574

[4] Birman, K.P., Hayden, M., Ozkasap, O., Xiao, Z., Budiu, M., Minsky, Y.: Bimodal multicast. ACM Transactions on Computer Systems 17(2) (May 1999) 41–88

[5] Stutzbach, D., Rejaie, R.: Understanding churn in peer-to-peer networks. In *Proceedings of the 6th ACM SIGCOMM conference on Internet measurement (IMC'06), New York, NY, USA, ACM* (2006) 189–202

[6] Saroiu, S., Gummadi, P.K., Gribble, S.D.: Measuring and analyzing the characteristics of Napster and Gnutella hosts. Multimedia Systems Journal 9(2) (August 2003) 170–184

[7] Mark Jelasity , Rachid Guerraoui , Anne-Marie Kermarrec , and Maarten van Steen. 2004. The Peer Sampling Service: Experimental Evaluation of Unstructured Gossip-Based Implementations. *Proceedings of the 5th ACM/IFIP/USENIX international conference on Middleware.*

[8] Chinmoy Dutta, Gopal Pandurangan, Rajmohan Rajaraman, and Zhifeng Sun. 2011. Information spreading in dynamic networks.

[9] M. Babaioff, S. Dobzinski, S. Oren, and A. Zohar. On bitcoin and red balloons. In *Proceedings of Electronic Commerce*, 2012.

[10] Miguel Castro and Barbara Liskov. 1999. Practical byzantine fault tolerance. In *Proceedings of the 3rd Symposium on Operating Systems Design and Implementation*. 173–186.

[11] Fischer, M. J.; Lynch, N. A.; Paterson, M. S. (1985). Impossibility of distributed consensus with one faulty process. *Journal of the ACM*. 32 (2): 374–382.

[12] R. Canetti and T. Rabin. Optimal Asynchronous Byzantine Agreement. Technical Report 92-15, Dept. of Computer Science, Hebrew University, 1992

[13] Christian Decker, Roger Wattenhofer. Information propagation in the bitcoin network. *University of California at Santa Barbara.* 2013.

[14] G.O. Karame, E. Androulaki, and S. Capkun. *Two bitcoins at the price of one? double-spending attacks on fast payments in bitcoin. In Proc. of Conference on Computer and Communication Security, 2012.*

[15] Tobias Bamert, Christian Decker, Lennart Elsen, Samuel Welten, and Roger Wattenhofer. *Have a snack, pay with bitcoin. In IEEE Internation Conference on Peer-to-Peer Computing (P2P), Trento, Italy, 2013.*